Stanford University

# Deep Learning-Based Collaborative Filtering

Omar Alhadlaq and Arjun Kunna

## Problem

Recommendation systems are of great interest to online services like Amazon, Netflix, and Spotify, as they derive a significant amount of revenue by accurately suggesting products that their users might enjoy.

However, non-deep learning methods have faced problems in dealing with matrix sparsity The literature shows that deep learning has much to contribute to this area, because it is able to effectively capture non-linear and complex user-item relationships.

In this project, we built a model to improve predictions for user-submitted ratings in recommendation systems by implementing a deep autoencoder model.

## Data

We are using the Netflix Dataset. This is a publicly available dataset with about 480k users and 100m ratings over 17770 movies. We had to adapt the data's original form into a form that was suitable to train a model on. The dataset was provided as a repository of 17770 sections, one per movie. The first line of each section contained the movie id with each subsequent line in the section corresponds to a rating from a customer and its date in the following format: (CustomerID, Rating, Date).

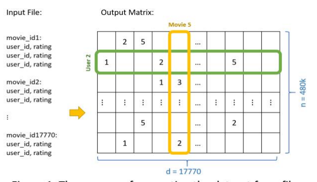

Figure 1: The process of converting the dataset from files to a matrix

For our model, we decided to represent each user by a vector, with each vector entry corresponding to a rating for a particular movie. Thus, each vector is in $\mathbb{R}^{17770}$. We wrote some scripts to parse the data files into this form and then wrote it back to disk in a format such that instead of being categorized by movie, it was categorized by user.

Thus, we ended up with a dataset as a matrix of dimension num movies × num users = 17,770 × 480,000 as shown in figure 1.

## Model

**AutoEncoder:**
An autoencoder is a network that consists of two transformations:
1) encoder(x) : $\mathbb{R}^d \rightarrow \mathbb{R}^e$
2) decoder(x) : $\mathbb{R}^e \rightarrow \mathbb{R}^d$

The goal of the autoencoder is to obtain an e-dimensional representation of the data, such that the error between X and f(x) = decode(encode(x)) is minimized.

The forward propagation equations for a two-layer autoencoder are outlined in (eq. 1). The input vector $X \in \mathbb{R}^{d \times 32}$, where d is the number of movies and 32 is the batch size. Each layer of the encoder and the decoder parts of our model consist of a fully connected neural networks computing $A = g(WX + b)$.

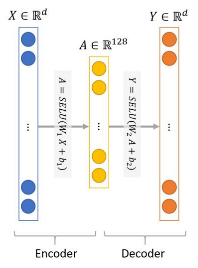

Figure 2: Baseline model

**Loss Function:**
We used a Masked Mean Squared Error loss function (eq. 2).

$$\begin{aligned} Z_1 &= W_1X + b_1 \\ A_1 &= SELU(Z_1) \\ Z_2 &= W_2A_1 + b2 \\ Y &= SELU(Z_2) \end{aligned}$$

Equation 1: Forward propagation of autoencoder.

$$RMSE = \sqrt{MMSE} = \sqrt{\frac{\sum_{i=1}^{n} m_i * (r_i - y_i)^2}{\sum_{i=1}^{n} m_i}}$$

Equation 2: the loss function.

$r_i$ is the actual rating, $y_i$ is the predicted rating, and $m_i$ is the mask function: $m_i = 1$ if $r_i \neq 0$, and $m_i = 0$ otherwise. Thus, we are only computing the loss on examples where we have the actual rating.

**Hyperparameters:**
1) Activation function:
According to a paper by Kuchaiev and Ginsburg, activation functions with nonzero negative part and unbounded positive part work best for autoencoders. Of these, SELU performed better than LRELU. Thus, we decided to use this as our activation function.
2) We used a momentum gradient descent with a momentum of 0.9, learning rate 0.005, batch size of 32.

## Results and Extensions

**Baseline Model:**
We implemented a basic model with only one layer for the encoder and decoder. This is the model depicted above in figure 2. We experimented with layers of size 128, 256, and 512 neurons. As depicted in figure 2, the training loss decreases as number of neurons increases. Additionally, we see that the model experiences overfitting as the number of neurons increases. The baseline model was able to achieve and RMSE of **1.084**.

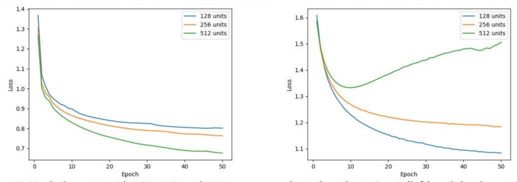

Figure 3: Single-layer AE with 128, 256, and 512 neurons evaluated on the train set (left) and the dev set (right).

## Cont. Results and Extensions

**Ext1. Going Deeper:**
Having established a baseline model, we tried adding more layers to improve the fit. We built a model with 3 layers each in the encoder and decoder. The first, second, and third layers had 128, 256, and 256 neurons respectively. In doing so, we chose layers of a small enough dimensionality (128) to reduce overfitting. In this model, we find that the dev error is lower than the shallow baseline model, as it reaches **0.951**.

**Ext2. Adding Constraints:**
In the basic model, we trained the weights of the encoder separately from that of the decoder. However, as the decoder is theoretically the inverse of the encoder, it is reasonable to constrain the decoder's weights $W_d$ to be equal the transpose of the encoders weights $W_e$. Thus, $W_d = W_e^T$.

This has the effect of effectively halving the number of parameters, and would be expected to reduce overfitting. However, in practice it appeared to have negative effects on both the training and dev sets, and increased the RMSE to **0.984**.

**Ext3. Adding Dropout:**
We also mitigate overfitting by implementing dropout. This acts as a form of regularization. We only applied dropout on the encoder. We experimented with dropout probabilities between 0.2 and 0.8 As seen in figure 4, low levels of dropout worked the best, with 0.2 resulting in the lowest dev error with an RMSE of **0.939**.

**Ext4. Dense Refeeding:**
Ideally, we would want the output of the autoencoder, f(x), to be dense to predict ratings of all the movies. Also we want f(x) = x for any x. To explicitly enforce these two constraints, we augment every optimization iteration with an iterative dense re-feeding step. We feed the AE output f(x) back to the model and perform the weight update.

We implemented this, however it did not improve the results as the dev loss was about **0.947**.

We ran our best model on the test set and got RMSE = **<u>0.943</u>**

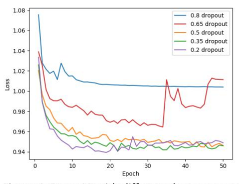

Figure 4: Deep AE with different dropout rates.

## Discussion

In this project, we have found that deep learning is able to aid recommendation systems in a meaningful way. It was a fantastic learning experience to have built a deep learning system from scratch, as we were forced to think about data-wrangling and pipeline questions as well. In this, project we got respectful results compared to other state-of-the-art models as shown in the table:

| I-AR | U-AR | RNN | URec | Our model |
|---|---|---|---|---|
| 0. 936 | 0. 965 | 0. 922 | 0. 910 | **0.943** |

Table1: Test RMSE of different models on the Netflix dataset.

Some of the surprising results we found were that constraining did not improve overfitting that much, and that dense re-feeding was not as effective as suggested by the literature. However, it was encouraging to note that going 'deeper' and dropout worked as the theory predicted. Deep learning has revolutionized many areas of machine learning and we are optimistic that it will be able to impact recommendation systems as well.

## Future Work

We relied on literature/canonical results for the optimal activation function and hyperparameters. It would be instructive for us to test this ourselves, if time permitted.